Spring 5-1-2002

The Neem Platform: an Evolvable Framework for Perceptual Collaborative Applications ; CU-CS-936-02

Paulo Barthelmess
University of Colorado Boulder

Clarence A. Ellis
University of Colorado Boulder

The Neem Platform: an Evolvable Framework for Perceptual Collaborative Applications

P. Barthelmess and C.A. Ellis

Department of Computer Science, University of Colorado at Boulder, Campus Box 430, Boulder, CO 80309-0430, USA. {barthelm,skip}@colorado.edu

Abstract. The Neem Platform is a research test bed for Project Neem, concerned with the development of socially and culturally aware collaborative systems in a wide range of domains, through development of situated applications that target specific group cultures and alternative theories. The Neem Platform is a generic (application neutral) component-based framework that provides functionality that facilitates building such collaborative applications.

Given the wide variety of target applications, the platform provides more than a closed set of behaviors - it is itself evolvable. Its functionality can be extended and adapted through facilities provided by a coordination mechanism that isolates computation from coordination concerns. Application development on top of this platform is facilitated by reuse of common functionality, as well as from embedded support for Wizard of Oz experiments.

A novelty of Project Neem is its focus on the dynamic aspects that characterize group interaction under a Perceptual Interface paradigm. Participants' multimodal interactions such as voice exchanges, textual messages, widget operations and eventually gestures, eye gaze and facial expressions are reified and made available to applications, that apply situated reasoning, using this rich contextual information to dynamically adapt their behavior. The system presents itself multimodally as well, through a set of *virtual participants* - automated entities that are perceived by human participants as having personalities and emotions, making use of animation and voice generation.

1 Introduction

Project Neem at University of Colorado is concerned with the development of socially and culturally aware collaborative systems. The project explores issues surrounding such systems in a variety of different domains, targeting different group cultures and interaction styles. These applications range, for instance, from very unstructured "extreme collaborations" [32] to rigid protocol-based interactions, such as the ones regulated by Robert's Rules of Order [44].

The Neem Platform is a research test bed for this project. It provides a generic (application neutral) evolvable framework upon which socially and culturally aware applications are developed. In the present work, we concentrate on describing the platform and only hint at the complexity and challenges involved in building such applications, which are explored more in depth elsewhere [18].

Central to the design of the platform is the focus of the project on the dynamic nature of group work in general, and social and cultural aspects in particular. The platform is

therefore designed to make available to applications rich contextual information and provide mechanisms to allow analysis and reasoning based on it, so that applications can elicit timely and appropriate responses that conform to situated social and cultural norms of a target group. The platform supports and incentives application designs that are based on dynamic, context appropriate system reaction, situated to specific groups of users.

Interaction between applications and users is supported under a *perceptual interface* [51] paradigm. Perceptual Interfaces are based on how humans communicate among themselves, exploring verbal and nonverbal human behaviors through the incorporation of multiple modalities, such as speech, gestures, gaze, both for collection and for presentation. Perceptual Interfaces take into consideration psychosocial aspects and aim at presenting a system in such a way as to make it an acceptable social actor.

The platform provides facilities for the integration of functionality for capture and reification of user actions over a variety of modalities (e.g. speech, text, gestures), as well as analysis and reasoning based on this reified context. Also included are facilities that allow applications to present themselves as a set of virtual participants, perceived as having personalities and opinions, through the use of multiple output modalities such as animated characters that emote, voice, natural language textual output as well as conventional widget based communication.

1.1 Platform characteristics

The goal of the platform is to facilitate the development of real-time distributed multipoint perceptual-based applications that focus on context-appropriate dynamic reactions. To this end, an evolvable component-based infrastructure is provided. Functionality embedded in the platform makes development of group-aware perceptual applications roughly equivalent to the development of singleware (single user applications). Development is also supported by embedded support for Wizard of Oz experiments. Wizard of Oz is a traditional technique for testing new features in the field by having a human participant masquerade as a virtual one, thus allowing for faster development cycles than possible if everything needs to be coded. This technique is usually employed by natural language based systems. Here we extend this use to explore issues on group interaction.

The platform incorporates reusable components that provide extensible support for basic real-time collaboration, such as session management, consistency control, data distribution, communication and coordination among components [22]. Its focus is on functionality that allows user actions to be captured, reasoned upon, and reacted to, under a perceptual perspective that explores multiple input and output modalities.

The burden of dealing with the above mentioned aspects is taken from the applications, that are free to concentrate on domain specific requirements. Development cycle is thus potentially reduced and made amenable to a rapid prototyping approach. Wizard of Oz experiments can also be easily implemented through simple tools, reducing the need for coding of highly experimental designs at too early a stage.

Given the varied and unanticipated requirements that are bound to surface during development of the wide range of proposed applications, the platform provides more

than a closed set of pre-existing behaviors - the platform itself is designed to be extensible and adaptable, and its functionality is expected to follow an evolutive cycle alongside applications. The distinction between application development and platform refinement is thus blurred.

The platform provides at its core an architectural coordination model that provides and environment for plugging-in distributed and potentially heterogeneous components. Computation and coordination concerns are handled separately, to enhance reusability. Computation is provided by message-enabled components. Coordination is mediated by a brokering mechanism, that offers functionality to "glue" components together by manipulating messages. The decoupling of coordination from the computation results in an architecture that promotes agile and flexible composition of behaviors, that can be made to fit a variety of styles.

We frame the extensibility problem in terms of aspect-oriented programming (AOP) [20] and we show that the coordination model adopted in Neem provides potential *join points* and the mechanism to transparently combine separately developed aspects, the hallmarks of AOP [19]. *Join points* places were independently developed aspects can we weaved together implicitly, without the writers of the code becoming aware of additional concerns. We also discuss advantages of using such approach in CSCW systems (Section 4.3).

Actual implementation of the platform is based on aggressive reuse of existing tools, standards and technology. Services and functionalities are provided to the maximum extent possible by off-the-shelf and open source solutions, that are integrated through wrappers. The immediate advantages are the reuse of robust software, availability of developers trained in the use of the tools, and possibility of replacing components with newer and improved technology as it becomes available.

1.2 Neem applications

Neem applications are built on top of the platform and provide those components that are unique to an application. A business meeting application can, e.g., employ interface elements that support creating an agenda, and tracking topics as they are discussed by a group. At the same time, such a meeting application might include active functionality that e.g. issues warnings whenever the time allocated for a topic is exceeded by some percentage. In this case, either a visual warning can be employed, or a warning can be channeled through a virtual participant.

Clearly, application specific interface elements and augmentation functionality are highly dependent on cultural and social aspects of a group of users. Even for the same domain, such as business meetings, what is appropriate support depends on the type of meeting that is being targeted (formal, informal), how groups are organized (e.g. hierarchy) and a multitude of social and cultural rules that determine what is to be expected from each participant, including the system, that must adhere to the social conventions of the group.

The needs at the application level for cultural adaptation is addressed in Neem by having generic (and evolvable) functionality provided by a platform upon which specific applications can be developed for each different kind of domain. The platform thus

offers a mechanism for reuse of generic functionality and supports rapid application development by encapsulating common CSCW aspects.
We envision employing the Neem Platform to develop end applications in different areas:

- Business meeting support - many different meeting models exist, that support informal to formal group collaboration scenarios. Each model would correspond to a different application layer built on top of the platform, each of them exploring alternative, perhaps conflicting social theories.
- Distance education - the rich environment potentially provided by perceptual-based user interfaces can be employed to support learning applications, where the students learning styles are accounted for.
- Universal access - the ease of integration of devices makes it possible in principle to support a larger population of users, that rely on specific modalities for information production or consumption (haptik, speech, for blind users, gesture based for deaf, and so on).

1.3 Organization of the paper

In the rest of this paper, we describe in further detail the functionality of the Neem Platform. We begin by presenting Related Work (Section 2), followed by the presentation of features of Perceptual Interfaces and the connection to multimedia and multimodal technology (Section 3). Section 4 overviews the architecture of the platform. The paper ends with Summary and Future Work (Section 5) and References.

2 Related work

Related areas (CSCW toolkits, perceptual interface based systems, architectures and coordination mechanisms) are too broad to be covered in a single section (or even a single paper!). We therefore highlight differences and similarities between our approach and that of others, illustrating with references to representative work by no means exhaustive.

The need for flexibility in CSCW toolkits and systems is well acknowledged (e.g. by [16, 22, 48, 24]. The common lesson is that flexibility requirements surface due to the varied and dynamic nature of group work, which renders approaches based on a closed set of behaviors useless, because of the restrictions they impose due to their closed vocabularies.

Two lines of work pertain to adaptable CSCW infrastructures: 1)“tailorable” systems, usually meaning that extensions are implemented by end-users (e.g. in [31, 48, 29]) and 2)work on extensibility by developers (e.g. [15, 17, 45, 46]. The Neem Platform follows the latter approach - it is meant to be adapted and extended by developers following a user-centered approach. We heretofore concentrate on discussing only this approach.

Flexible CSCW toolkits differ in the functionality they target and therefore in the kinds of application aspects that they facilitate. As an example of this variety, consider:

Prospero [15, 14] which concentrates on distributed data management and consistency control mechanisms; Intermezzo [17] focus on the coordination aspects of collaboration, in support of fluid interactions, offering user awareness, session management, and policy control; GroupKit [45] offers a basic infrastructure that includes distributed process coordination, groupware widgets and session management.

The Neem Platform offers a generic coordination infrastructure that could in principle be used to build virtually any kind of system, but the functionality that is built on top of this generic layer targets real-time distributed multipoint dynamic collaborative systems based on a perceptual interface paradigm. Participants multimodal actions are reified and this context is made available for analysis and reasoning. Facilities for multimodal presentation through virtual participants is also offered.

Analysis of a broader range of toolkits under the perspective of functionality and flexibility can be found e.g. in [16, 22, 13].

From a Perceptual Interface research perspective, Neem differs from typical work because of its focus on group interaction, as opposed to the focus on single users, as is typical in the area. Even in systems that target groups of users (e.g. [8, 34, 41]), the focus is on multimodal command, in which speech and pen, for instance are used to replace more conventional interface devices.

Neem employs an opportunistic approach where the system dynamically adapts based on reasoning over a context of (mostly human-to-human) interaction, as opposed to receiving direct (multimodal) commands from individual users. In this sense Neem is more closely related to the work presented e.g. by: Jebara et al [26] in which the system acts as a mediator of the group meeting, offering feedback and relevant questions to stimulate further conversation; Isbister et al [25], whose prototype mimics a party host, trying to find a safe common topic for guests whose conversation has lagged; Nishimoto et al [36] whose agent enhances the creative aspects of the conversations by entering them as an equal participant with the human participants and keeping the conversation lively; CMU's Janus project [40] is somewhat related, in its aim to make human-to-human communication across language barriers easier through multilingual translation of multi-party conversations and access of databases to automatically provide additional information (such as train schedules or city maps to the user) [54]. While Neem shares the interest in human-to-human mediation, its goals are more ambitious than keeping a bi-party conversation going. Neem targets social and cultural aspects and is therefore concerned with a more detailed view of how groups work, and how collaborative systems can contribute.

From the perspective of extensibility, different strategies are employed by CSCW toolkits: in Prospero [15], extensibility derives from a reflective mechanism that makes use of facilities provided by the host language employed - CLOS. The guiding paradigm is that of Open Implementation [28], that proposes gaining flexibility by breaking the encapsulation that is traditional in object oriented development, making them amenable to meta-level control mechanisms. Intermezzo's allow code (written in an extended version of Python) to be dynamically downloaded, and executed at the time it is downloaded or as a response to object and database change events described via a pattern matching language; Groupkit [45] supports programatic reaction not only to system generated events, but also to application specific ones. Events can trigger application

notifiers or handlers. Particularly, such notifiers can be associated with changes to *environments* - dictionary-style shared data models - that the system automatically keeps consistent across replicas.

Flexibility and extensibility in Neem are result from its foundation on a core architectural *coordination model*. In this model, decoupled components interact indirectly through message exchanges. A meta-level mechanism mediates access to a Linda-like tuple-space [4]. This functionality, that we call *mediation*, allows for explicit control over component cooperation through message-rewrite rules.

The approach is related to data-centered architectures and is therefore related to some extent to a great number of similar approaches (see [38] for a comprehensive survey). Neem differs from these approaches by providing an explicit locus of (meta-level) coordination control, the *mediation* functionality mentioned above.

More specifically, Neem shares Laura's view of components as providers of multiple services [50]. Unlike Laura, there is not a necessary one-to-one correspondence between request and provision of services. Requests in Neem can be serviced in parallel by multiple components.

As in Gamma [2], LO [1] and derivatives (e.g. COOLL [5] and ShaDE [6]), Neem's explicit control mechanism is based on multiset message rewriting. Unlike these models, Neem provides a meta-level mechanism that executes transformations according to a reified *composition policy*. This mechanism offers control over the "glue" that binds components together, and provide for recombinations to be performed without components knowledge and thus without the need for component recoding.

Neem's approach to coordination can be also related to some multi-agent platforms, such as the Open Agent Architecture (OAA) [33] and to some extent to the Galaxy Communicator Architecture [11], both of which could be classified as data-centered as well. In these platforms, as in Neem, coordination is mediated by active elements - *Facilitators* in OAA and the *Hub* in Galaxy.

3 Perceptual Interfaces

Perceptual Interfaces are based on a paradigmatic shift from the structured, command based GUI interfaces to a more natural one based on how humans interact among themselves. These new kinds of interfaces have been extensively studied, among others by Reeves and Nass at Stanford's Center for the Study of Language and Information (e.g. [43, 42]). Their findings support the notion that provided that an interface mimic real life, even if imperfectly, principles that explain perception in real life can be applied straightforwardly to computers, i.e. that people's reactions to computers are fundamentally social and perceptual [43]. The reaction to animated characters, for instance, tend to be similar to real participants, and even gender, ethnicity and similar factors play similar roles independently of the obvious artificial nature of such characters, and their imperfections in movements, voice.

The use of such a paradigm seems particularly relevant and appropriate in the context of group collaboration. The bulk of communication is already performed by humans among themselves. The objective of a group system is in fact to support such human to human interaction. It is therefore natural to employ a similar communications-based

paradigm to integrate augmentation functionality in a seamless and transparent way. It is thus desirable on one hand for a system to extract information from an ongoing interaction among humans, rather than by direct commands given through conventional GUIs, and on the other hand to present system's contributions through similar mechanisms as the ones employed by human participants, i.e., through a complex combination of speech, gestures, facial expressions, gaze, etc.

Perceptual Interfaces are associated to other technology, particularly multimedia and multimodal, that can be related to each other to form a 4-tiered structure (Figure 1). *Multiple media* channels (e.g. audio, video) broaden human perception of others. Information carried over these channels are analyzed according to multiple *modalities* (e.g., natural language from voice and text, gestures, prosody, facial expressions, gaze). This rich information is then used to build context and awareness of the interaction at the *perceptual* level. This context allows the system to elicit reactions that are grounded on psychosocial aspects, allowing the system to be perceived as a meaningful social actor on its own right [43]. While we concentrate on the present work on discussing support for perceptual features (the first three levels we just described), social and culturally aware systems add a fourth layer that is concerned with the social dynamics of a group. Here we only hint at the issues that surround the development of such systems.

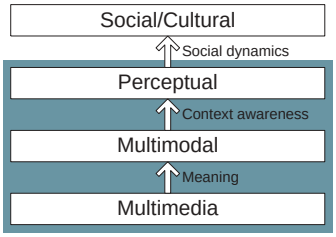

Fig. 1. Perceptual Interfaces.

The use of multiple communication channels (multimedia) has been extensively researched in the last decade and has resulted mainly in improved information presentation capabilities, through the addition of video, animation and sound to the interface. Multimodal systems take this concept one step beyond. A multimodal system is able to automatically model the content of the information at a high level of abstraction. A multimodal system strives for meaning [35].

Most common multimodal interfaces combine speech recognition and lipreading (e.g. in [39]), or speech and pen based interfaces (e.g. in [37]), but other combinations are also explored for instance: the integration of speech and gestures (e.g. [47]); of speech, eye-gaze and hand-gestures (e.g. [30]); face and gesture (e.g. [7]). It is not uncommon to find combination of speech and more conventional user interface modalities, such as keyboard and mouse related ones as well.

While individual modalities provide a wealth of information that is commonly not present in conventional interfaces, the combination of modalities provides still richer information. *Fusion* is the process that combines individual modality streams into a single

one, based on time and discourse constraints [27]. Fusion unleashes the full power of multimodal communication, allowing information on each mode to complement each other. A classic example of such combination was employed in the Bolt's pioneer system, 'Put-that-there' [3], that combined speech recognition with gesture analysis that allowed users to point to objects and locations and issue verbal commands to have them moved.

The combination of modalities during group interaction opens up possibilities for analysis never before available. It is for instance possible to combine facial expressions with voice analysis to determine if a user looked (or sounded) angry or happy while (or just before) issuing some utterance. Fusion has therefore the potential for adding context to the interaction that would otherwise go un-noticed.

If *fusion* combines different modes, *fission* does exactly the opposite. Given a message that needs to be conveyed, it is possible and desirable to pause and consider what is the most effective way of conveying it, either through some conventional user interface mechanism, or through a voice message, or through the use of an animated character that emotes. In other words, multimodal interfaces provide an opportunity for alternative renderings of the same information, so that it imparts the importance and content appropriately, taking into account social and cultural rules.

Fusion and fission mechanisms have a potential to liberate users from having to adapt to system mandated input and output mechanisms. Users can be free to choose the modality that best fits their styles both when producing information as well as when being presented with information produced by a system. There is therefore a potential for adaptation to user needs and styles. To fully realize user adaptation, besides having some form of user modeling, a system would have to provide support for translations between modes that may not be trivial. A communication between a visually oriented user (say a deaf person) with users that prefer to speak would require translation to and from a spoken language and a visual, gesture-based language that is not in the least trivial.

Perceptual group applications have, nonetheless, the potential for offering an integration framework for such technology, once it becomes available, thus making possible the development of universally accessible systems.

Figure 2 depicts information flow in Neem's Perceptual Interface. Participants interact in a distributed collaboration environment and their actions are *sensed* and *interpreted*; multiple interpreted streams are combined during *fusion*. *Reasoning* analyzes the events in context, i.e., it takes into consideration past interaction. A *response* is generated (which includes doing nothing), *fission* determines the most effective way to react given available modes and taking into consideration user characteristics. This potentially complex sequence of multiple modality actions are finally rendered at one or more participants stations (reaction).

While Perceptual Interfaces are not new, the use in a collaborative system for extraction, analysis and reasoning about human-to-human interactions in support of social and cultural awareness is unique to Neem.

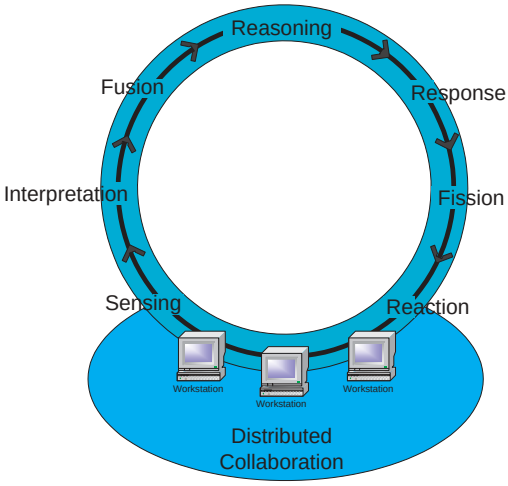

Fig. 2. Perceptual information processing.

4 Architecture

We now turn our attention to the platform's architecture. The platform's design is based on a 3-tiered approach:

1. The platform's foundation is a generic *coordination model* that can in principle be used to implement just about any distributed component-based system. It is at this core level that flexibility and extensibility are introduced into the platform.
2. A second stage specializes this generic infrastructure to support collaboration and dynamic reaction based on a perceptual paradigm, as is the goal of the project.
3. The third and final level realizes the collaboration framework as an actual implementation that makes extensive use of off-the-shelf and open source solutions.

We now discuss in turn each of these thee layers in further detail.

4.1 Coordination model

A *coordination model* "provides the means of integrating a number of possibly heterogeneous components by interfacing with each other in such a way that the collective set forms a single application that can execute on and take advantage of parallel and distributed systems" [38].

The advantage of clearly separating coordination from computation comes from the flexibility in rearranging computational elements to fit new styles without having to modify the components themselves, just the "glue" that binds them together, enhancing chances of reuse.

In the coordination model adopted by Neem, decoupled components interact indirectly through message exchanges. A meta-level mechanism mediates access to a

Linda-like tuple-space [4]. This functionality, that we call *mediation*, allows for explicit control over component cooperation through message-rewrite rules.

A tuple-space is a shared dataspace, through which cooperating processes communicate among themselves only indirectly, by posting or broadcasting information into the medium, and by retrieving information by removing or just copying information off the medium. Retrieval is based on content-addressing capabilities of the medium [38].

Messages generated by the components and channeled through mediators can be seen as events, some of which are interpreted as requests for service that will cause one or more components to be in turn activated thus establishing indirect communication between the component that generated the original message and the one(s) that got activated as a result of mediator internal actions (Figure 3). Components react to messages by performing some computation and eventually sending out one or more messages.

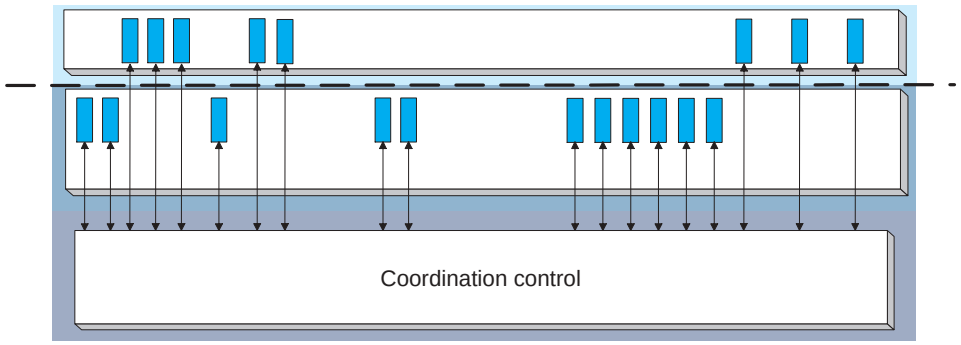

Fig. 3. Neem coordination model. Components communicate indirectly through a *coordination control* mechanism that embeds tuple-space functionality and a *mediator*. The dotted line indicates a separation of reusable components (below the line) from application specific ones (above the line. The distinction is purely based on reuse, and therefore is not clear-cut.

Changing communication aspects therefore implies exclusively in modifying mediator definitions to “rewire” component communication patterns, without having to touch any of the existing components. The writers of the components are in fact largely unaware of the actual contextual use of their components beyond required handling of messages that are received and responses that are generated. The ability to compose existing behaviors in unanticipated ways results in clear flexibility in their potential for reuse (for a short and clear comparison of different communication styles, see [49]).

Components are uniformly employed both for the development of core platform functionality and application layers. The only difference from a developer’s point of view is the potential for reusability of components at each of these levels - while platform components are expected to have general use, application specific components are tied to a specific solution and therefore have less chance of reuse. In the following paragraphs, we describe mostly generic functionality, that therefore corresponds to reusable platform elements, rather than application level ones.

We now define more precisely *messages*, *components*, *component behavior* and *mediators*, the basic elements of the coordination model.

Messages Given a set K of *keys* and V of *values*, a *message* M is a set of $(k, v), k \in K, v \in V$, i.e., messages are structured as frames. The set of *keys* of M is denoted by $k_M(M)$, and $v_M(k, M)$ denotes the (unique) *value* that corresponds to *key* k in message M.

A *pattern* P is a sentence in logic of the form $\lambda m.A_1, A_2, \ldots, A_n$, where m is a message that is matched against the pattern and $A_i, i = 1 \ldots n$ are atomic terms of the form $k \in k_M(m)$ or $v_M(k, m) = c$, $c \in V$ is a constant. A pattern P is said to *match* a message m if all its n terms are true for message m, i.e., if all the pattern's keys are in the message and their corresponding values, if specified, are the same.

Patterns can therefore be seen as partially instantiated messages/frames, in which some of the keys have assigned values and others correspond to "wildcards", i.e., they can match any values.

Components Components are the locus of computation and are characterized from the coordination model perspective by their signatures, given by the tuple $S = (O_M, I_M)$, where O_M and I_M are sets of *patterns* as specified above.

Each outbound message generated by a component must match one of the patterns in O_M. Conversely, I_M describes messages that are serviced by a components, i.e., component inbound messages must match a pattern in I_M. We will say that components *accept* messages that conform to I_M and *generate* messages according to O_M.

Component outbound messages serve a dual (and indistinguishable) purpose of signaling events and requesting services from other components. Inbound messages represent service activation requests originated elsewhere in a system.

Components can be *instantiated* by the coordination mechanism, meaning that a new process or thread is started on some machine. A component is said to be *active* if at least one instance is in execution at a given time. Each instance has its own private state, separate from that of other instances.

Multiple active components can have non-disjoint, or even identical signatures. In practical terms, that means that messages with similar structures (as defined by the patterns in components' signatures) might be both produced by multiple components as well as serviced by multiple components as well. In particular, multiple components might have identical signatures if they are instances of the same component. A single message is thus accepted by multiple active components, that process the corresponding service in parallel, for instance displaying information simultaneously in multiple user interfaces.

Component behavior can be understood from a "black-box perspective as transformations that map I_M to O_M, in the form of non-deterministic *rewrite rules*:

$R(m_I) = P_{I_1}, P_{I_2}, \ldots, P_{I_n} :- M_{O_1}, M_{O_2}, \ldots, M_{O_q}$

where n or q but not both are possibly equal to zero, m_I is a message that is to be matched against the rule, $P_{I_i}, i = 1 \ldots n, n > 0$ are *patterns* as defined above and $M_{O_j}, j = 1 \ldots q$ with q possibly equal to zero are messages specified as:

$M_{O_j} = \{(k_1, v_1), (k_2, v_2), \ldots, (k_q, v_q)\}$

where $k_i, i = 1 \ldots q$, $k_i \in K$ are *keys*, not necessarily in $k_M(m_I)$ and $v_i \in V$ is either a constant value or some value $v_M(k, m_I), k \in k_M(m_I)$, that is, the body of a rule builds messages by specifying a set of key and value pairs constructed from constants in the respective domains (K, V) or values extracted from the message m_I that is being matched against the rule.

Rules with empty bodies do not generate any messages, i.e., matching messages are simply consumed. Rules with empty heads indicate that messages are produced (non-deterministically) without the need for a matching input. This might happen if messages are generated as responses to component state changes triggered, for example, by user interface actions.

Messages m_I that match more than one rule head result in a non-deterministic firing of one of them. Again, the apparent non-deterministic behavior results from (hidden) component state changes that appear to an external observer (such as the coordination model) as being random.

Rewrite rules conceptually describe the observable behaviors of components in terms of messages consumed and generated. Component cooperation patterns result from the chaining of messages, as components react to messages generated by others and in turn generate messages that will cause further component activations. In Neem, these cooperation patterns can be modified without the need to recode components, through a reflective mechanism that we present next.

The mediator offers a mechanism through which control can be exerted over a system's cooperation pattern, by transparent message transformations. The mediator therefore offers control over the "glue" that binds components together, and provides for recombinations to be performed without components knowledge and thus without the need for component recoding.

The mechanism is based on the isolation of component writes and reads, that are (conceptually) done on two distinct tuple-spaces, that we will call α and β respectively. Components are required to write exclusively to the α-space and read exclusively from the β-space. The mediator bridges the α and the β spaces by executing mirrored reads and writes, i.e., by reading exclusively from the α-space and writing exclusively to the β-space (Figure 4).

A reified *composition policy* dictates transformations that are applied by the mediator - messages read by the mediator from the α-space are transformed before being written to the β-space. A *Composition policy* is a explicit script that lists *composition rules*, that we take to have the same form as the *rewrite rules* described above, with the understanding that here m_I is a message read from the α-space, that we denote in this context as m_α; the patterns P_{I_i} refer to α-message structures, that we denote as P_{α_i} and the generated messages M_{O_j} are written to the β-space, heretofore denoted as M_{α_j}.

The execution mechanism consists of matching α-messages to cooperation rules heads and generating β-messages according to matching rules' bodies. In the context of mediation, rule execution has a deterministic semantic: a single α-message that match multiple rules cause all of them fire; empty rule heads are not allowed; empty rule

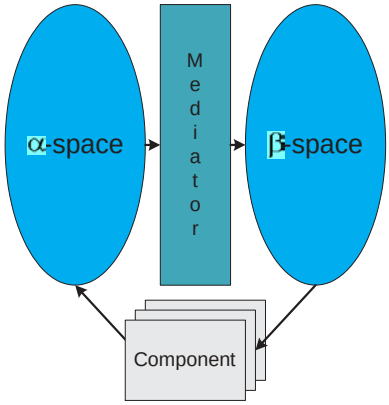

Fig. 4. Components and mediators apply reads and writes to complementary spaces α and β.

bodies consume messages, as usual; α-messages not matched by any rule are written to the β-space without transformations.

From an extensibility point of view, coordination rules provide for flexibility by allowing redirections and introduction of new service activation requests without the need to recode components, thus letting behaviors be recombined to suit different application styles.

Service activation All active component whose I_M-signatures match each of the β-messages generated by a rule will have the corresponding service activated. Since multiple active components can match the same messages, a single β-message can result in a broadcast, or more usually in a multicast. Unicasts, resulting from having a message match a unique component, are usually performed by including some identification value in a message, that corresponds to a unique pattern in some I_M-signature of an active component, e.g., by having a key "id" that uniquely identifies each component that needs to be addressed unequivocally.

4.2 An example

We now present an example that illustrates the functionality of the coordination model. We start by defining an application that uses a simple user interface components that we then modify to show the model's extensibility features. In the following paragraphs, key names in *patterns* and *composition rules* are prefixed by a "♢", to visually distinguish them from values.

Distributed user interface updates To illustrate how a distributed user interface can be kept updated, consider a simple tool that has one button bt and a textual field tf, that displays an identification of the participant that last clicked on the button. The goal in a distributed user interface is to keep all users aware of each others actions, here, button clicks.

A component CLICKER implements the shared interface. Its signature is
$S_{Clicker} = (I_M = \{(\diamondsuit Class, Clicker), \diamondsuit Id)\}, O_M = I_M)$.

CLICKER thus produces and accepts the same kind of messages. The key $\diamondsuit Class$ identifies the component's type; $\diamondsuit Id$ carries a user identification. Multiple instances of CLICKER can be active, one per user in a group. Unique $\diamondsuit Id$ values are assigned to each of these instances.

Whenever a user clicks on the button bt at her interface, the component instance posts a message to the α-space announcing this event, through a message that carries the identification of the user attached to $\diamondsuit Id$.

The mediator at this point has no rules concerning this kind of message, so every such message is copied unchanged to the β-space.

All active instance of CLICKER accept each of these β-messages because of the match with their I_M-signatures. All instances retrieve the message in parallel and provide the associated service, in the particular case, each instance updates the textual field tf by displaying the value attached to the $\diamondsuit Id$ of the message that was retrieved. All users get thus notified of every user (including their own) button clicks.

Contextual use of messages We now extend the application by introducing a component TALLY that illustrates how a message can be used in additional contexts without modifications of existing components. Suppose we want to add functionality that counts clicks from each user and generates a notification once a user clicks on bt a predefined number of times, say five. Users that clicked that many times have their id's displayed in a list kept by another interface component, HIT5, to be described momentarily.

TALLY has a signature
$S_{Tally} = (I_M = \{(\diamondsuit Class, Clicker), \diamondsuit Id\}, O_M = \{(\diamondsuit Class, Tally), \diamondsuit Id\})$

TALLY accepts messages that comply to CLICKER generated messages, but in a different context. TALLY keeps a count of individual user clicks. This count gets incremented each time a message generated by a user gets accepted (in parallel to being accepted by all other instances of CLICKER). When individual counts reach the threshold (five), TALLY generates a message announcing this event, formatted according to its O_M signature.

HIT5 is the interface component that displays users that reached the threshold. HIT5's signature is
$S_{Hit5} = (I_M = \{(\diamondsuit Class, Tally), \diamondsuit Id\}, O_M = \{\})$

HIT5 accepts messages that comply to TALLY's generated messages and upon retrieval of one such message, updates the user interface to include the user's id in a list that is displayed. The empty O_M-signature signifies that no messages are generated by HIT5.

Reuse of components with incompatible signatures We just saw how an application can be extended by adding functionality that is based on the same messages, that are interpreted in a different context. We now show how the *mediator* can be used to make possible the reuse of existing components whose signatures do not match desired messages.

Suppose we are interested in saving in a log the sequence of user clicks generated by CLICKER instances. We could of course write a a new component that accepts messages compatible with CLICKER's, but here we want to reuse an existing (generic) component that was not built with CLICKER's functionality in mind. Suppose that LOGGER is a component that already does the required logging, but has a signature

$S_{Logger} = (I_M = \{(\diamondsuit Class, Logger), \diamondsuit Event, \diamondsuit Parm\}, O_M = \{\})$

LOGGER saves whatever is informed in $\diamondsuit Event$ and $\diamondsuit Parm$ into a log. Since LOGGER's I_M-signature is different from that of CLICKER's, a transformation needs to be performed. We employ a mediator *composition rule* for that purpose:

$\{(\diamondsuit Class, Clicker), \diamondsuit Id\} : -$
$\{(\diamondsuit Class, Clicker), (\diamondsuit Id, v(\diamondsuit Id))\},$
$\{(\diamondsuit Class, Logger), (\diamondsuit Event, Clicker), (\diamondsuit Parm, v(\diamondsuit Id))\}$

This rule's head matches CLICKER generated messages and its body causes two messages to be written to the β-space: 1) a copy of the original message, so that components that depend on this message will continue to have access to it, and 2) a message that matches LOGGER's I_M-signature. In the rule above, $v(\diamondsuit Id)$ is shorthand for $v_M(\diamondsuit Id, m_\alpha)$ given that a rule is always matched against a single message m_α at a time.

If we want to add logging of TALLY events as well, we can modify the rule to make it more generic:

$\{(\diamondsuit Class, Clicker), \diamondsuit Id\},$
$\{(\diamondsuit Class, Tally), \diamondsuit Id\} : -$
$\{(\diamondsuit Class, v(\diamondsuit Class)), (\diamondsuit Id, v(\diamondsuit Id))\},$
$\{(\diamondsuit Class, Logger), (\diamondsuit Event, v(\diamondsuit Class)), (\diamondsuit Parm, v(\diamondsuit Id))\}$

Now both CLICKER and TALLY notifications are included in the log. Notice that we made the bodies more generic by replacing constants with keys.

4.3 Design style

Before proceeding to the next architectural layer, we discuss issues related to design style. Component reusability depends on more than the existence of a coordination model. Clearly, depending on the services that are provided by components, and depending on the decomposition that is chosen, components might need to be recoded and modified when some extensions are introduced. We here discuss two factors that impact reusability, the use style that is embedded in *protocols* and *decomposition criteria*.

Message protocols The enhancements that we introduced in the example depend on the existence of a protocol that has to be followed by components. The ability of components to cooperate depends on availability or required information, that must be embedded in messages, not only for processing, but also for message routing.

While we do not dictate a specific message content and services, and the coordination model itself does not make any assumptions about that, we found useful to structure messages according to the following:

– We require components to send out messages announcing they have just been instantiated (*alive* message) and that they are about to terminate (*dying* message). Components that require detailed knowledge about which actual instances are active at any time make use of these messages. The messages inform class and unique identifier of instances.
– Components are also required to respond to *shutdown* messages by terminating, to allow remote deactivation. User interface components are required in addition to respond to *hide* and *show* messages, by making themselves invisible and visible respectively.
– All messages are required to identify the component that generated the message by including class and instance identification attached to $\diamondsuit Class$ and $\diamondsuit Id$ respectively. Class identifiers are useful in multicasts, while unique instance identifiers are useful for unicasts. We require all components to be directly addressable through this field by declaring I_M patterns that that are uniquely matched by each component instance. Every service offered by a component should be accessible through one such "unicast" pattern.
– Since message activation is asynchronous. it is useful to have component services announce service completion through a message. Synchronous behavior can then be emulated by watching for these completion notifications. This kind of messages are useful for meta-level components, that do other types of synchronization, for instance, wait for all instances of a service to terminate and then initiate some other action, for instance activating a second service that is constrained to start only after a first service terminates, for example.

(De)composition - an Aspect-oriented view Components that embed assumptions about aspects that are bound to change are subject to potential need for recoding. Take for instance a component that is built on the assumption of centralized data distribution - it cannot be reused in applications that require distributed replicas, for example.

This is an old problem, addressed by techniques that are based on separation of concerns. One such technique - Aspect-Oriented Programming (AOP) - proposes a style of programming that goes beyond object-orientation, in the sense that it tackles the problem of *cross-cutting* concerns. Cross-cutting concerns are those that cannot be clearly localized within a structural piece of code, for instance a class in an object-oriented system. Typical examples of cross-cutting concerns include coordination itself, and e.g. security and auditing, which are usually spread throughout multiple code units, making maintenance harder.

A variety of techniques, mainly programming language related, address this issue in different ways (see [20] for a comprehensive overview). What these techniques have in common is that they offer mechanisms that make possible to code aspects in isolation and then transparently recombine these different related behaviors automatically into a single larger system [19]. This mechanism is based on 1) specification of *join points*, places where aspects need to be weaved together and 2) transparent invocation of additional behaviors that correspond to the separately defined aspects at these points.

In the model we are describing, each α-message processed by a mediator implicitly offers a potential *join point*. Composition rules triggered by α-messages can activate

functionality that relates to multiple aspects, by generating multiple β-messages that cause components related to aspects to be transparently activated. Selected messages can for instance trigger the activation of encryption services, start access control operations and data distribution services where these different aspects, implemented as one or more components, can be applied transparently to code that is unaware of these different repercussions. This allows for aspects to be added, replaced or evolved without affecting existing functionality.

In our example, we showed how logging can be transparently added to an application, thus weaving in a concern that was not originally considered by the application. Similarly, other concerns can be added without the need for recoding existing components.

AOP is particularly appropriate in the context of collaborative systems, given identified common themes and aspects, such as consistency control, data distribution, concurrency control, access control, and fault tolerance for example. The literature of the area acknowledges both the importance and pervasiveness of these aspects, as well as the varied ways in which they have to be dealt with in the context of collaboration, depending on the target applications (and possibly even within a single application).

4.4 Framework

In the previous section, we examined the foundation of the platform - the coordination model. This model, while flexible, offers no specific support for the perceptual interface based dynamic collaborative applications that are targeted by Project Neem. In this section, we examine how the generic foundation is specialized to offer necessary services.

Given Neem's focus on real-time distributed multipoint perceptual interface-based applications, the framework embeds the following functionality:

- Multimodal input and output - extraction and presentation of multiple modality information.
- Multimodal processing - such as fusion, fission, parsing of natural language streams.
- Multimedia processing - audio mixing and video switching.
- Session management - creation, joining, leaving, meetings or sessions.

Application development support covers the areas:

- User interface components - for building shared artifacts.
- Support for reasoning - for implementing dynamic context-based reaction.
- Support for Wizard of Oz experiments - to allow experimentation without coding.

We start to address these issues by specializing components into *interface* and *augmentation* components. Even though conceptually similar (both are message enabled component types), *Neem Interface Components* (NICs) are characterized by their attachment to one or more interface devices, which makes them suitable for collecting and relaying interface events generated by each participant, in the form of standard messages. A *Neem Augmentation Component* (NAC), on the other hand, does not have this constraint and is purely a message processing device.

NICs provide means for the integration of multimedia devices, such as conventional monitor, keyboard, mouse, consoles, audio and video. Other less conventional devices (e.g. Virtual Reality (VR) goggles, haptik devices) can also be integrate through NICs. All that is required to integrate a new device is a set of NICs that interface with a device, extract events commanded by users and modify its state (for devices with output capabilities) according to commands received as messages. A NIC may for instance attach to an audio source (e.g. microphone) and do speech-to-text conversion, or extraction of prosodic features, or attach to a video source and do gesture or facial expression extraction. NICs also react to messages they receive, causing changes to the associated state of the interface, for instance rendering at users stations of textual messages, graphics or full animations including gesture and/or voice.

A single NIC can attach and service multiple devices, as is typical, for instance, in conventional GUI-based ones, where a single component attaches to a video monitor, keyboard and mouse. Conversely, one device can be tapped by multiple NICs. Information from a video source can for example be extracted by a set of NICs, each specializing in one modality, e.g. facial expressions, or gestures.

Wizard of Oz functionality is supported straightforwardly by NICs that offer an interface through which a human participant can activate the generation of messages that cause other components to react. One can, for instance, send messages to components that control animated characters, making them move, speak, emote, and so on. Similarly, any other component can be made to react by issuing appropriate messages from a wizard interface.

Neem Augmentation Components (NACs) provide mostly back-end functionality, i.e., they are mostly responsible for processing the multiple modality streams, e.g. parsing natural language streams, *fusion, fission* of different streams and so on, as well as providing session management and multimedia processing capabilities.

NACs are also responsible for providing support for reasoning about the perceived context of an ongoing interaction and generating appropriate responses. Responses themselves are dependent on the specific application that is built on top of the platform.

NACs typically collaborate on refining messages. Some NACs receive and process messages that represent participants actions directly. These NACs typically apply an initial transformation that is further refined by other NACs, obeying a cycle depicted in Figure 2. At the end of the cycle, one or more responses might have been generated. Responses are implemented as messages, that take effect as NICs react causing changes to one or more participants interfaces.

In actual use, multiple instances of the same NICs are usually active, normally one instance per participant. Each participant's audio channel, for instance, has a speech-to-text NIC attached to it, that is responsible for transcribing that participant's utterances. Some of these instances will correspond to NICs that perform generic (platform) functions, such as multimodal i/o, while others will correspond to shared artifacts through which a group interacts.

NACs, on the other hand, tend to be instantiated just once, and usually provide services to many (or all) participants. NACs offer therefore a convenient way to factor out common functionality. While NICs are necessarily deployed on participants' stations, NACs can be deployed anywhere (including on participants' stations), due to the

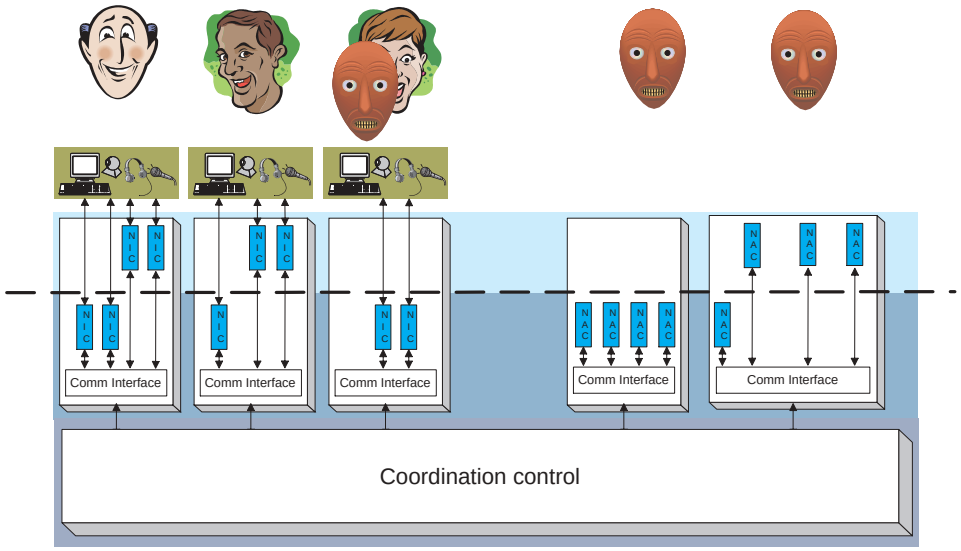

Fig. 5. Neem component deployment. Boxes surrounding components represent machines.

distributed nature of the multi-agent environment that supports intercommunication. Because of the heavy weight processing associated with multimodal processing, and for ease of administration, it is convenient to deploy NACs in one or more server machines, rather than on users' stations. Figure 5 shows a deployment mapping of NICs and NACs.

4.5 Prototype

A prototype of the platform has been implemented. Actual development leverages as much as possible on existing, field tested technology, based on open standards (Figure 6).

The reuse of existing technology leads to a rearrangement of components given that some single packages, such as the Multipoint Control Unit (MCU) embeds functionality that we mapped to multiple NACs in the framework, in this case, *session management* and *multimedia processing* functionality.

A messaging infrastructure is implemented as two distinct environments - a *collaboration* and a *multi-agent* environment that are connected through a *coupler* component. The *distributed collaboration environment* provides support for participants' interaction through NICs and the *multi-agent environment* supports back end augmentation functionality, such as multimodal processing and reasoning, which are typically provided by NACs. The reason for this split has to do with the different communication patterns that are typical of NICs and NACs. The *collaboration environment* offers the broadcast and multicast that are required by many NICs, while the *multi-agent environment* better supports that cycle of incremental refinements that characterize multimodal processing (see Figure 2). In this cycle, components typically apply a partial transformation,

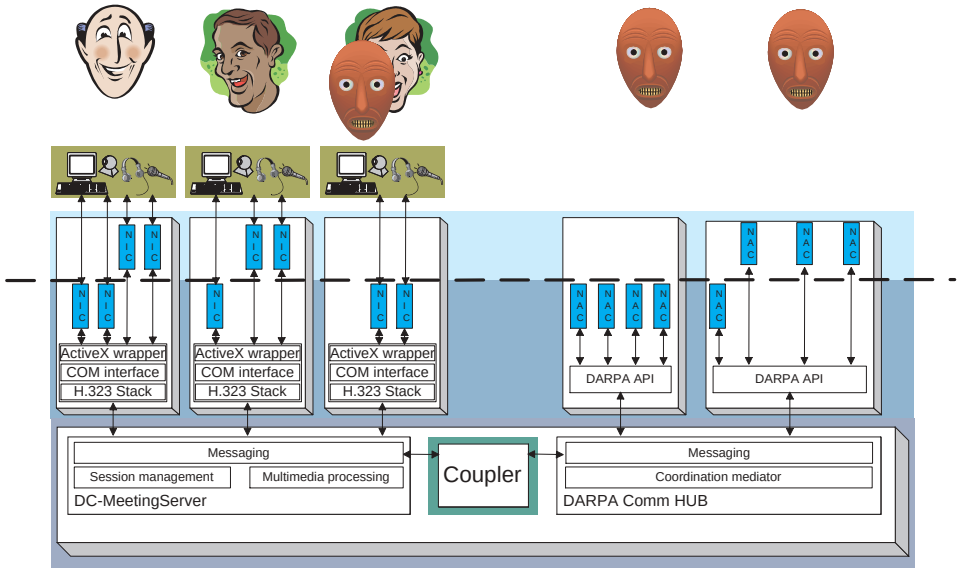

Fig. 6. The Neem prototype. Some macro-level components (MCU, DARPA Comm) incorporate multiple functionality.

that is posted back to the environment for further processing by other components, in a pipeline fashion.

Distributed collaboration environment Provides message delivery to groups of distributed participants (or rather to the NICs through which they interact), either through broadcasts or selective delivery. This environment is implemented by DC-Meeting Server and also embeds support for *session management* and *multimedia processing*.

DC-MeetingServer is a commercial H.323 Multipoint Conference Unit (MCU), produced by Data Connection Limited [12]. H.323 is a family of multimedia conferencing protocols published by the International Telecommunication Union [52]. These protocols establish a set of services that can be employed as a basic multimedia conferencing support layer. It includes, among others, services for conference creation, handling client connection and disconnection, file transfer, white-board, application sharing, video and audio communication and transfer of data between two or more connected clients. Also included is support for remote launching of applications in selected connected clients. A variety of server and client software based on this protocol is readily available in many platforms.

Multi-agent environment This environment is organized in a hub-and-spoke configuration. The *mediator* (hub) controls the flow of information among other components (spokes). The mediator keeps a state that can dynamically influence the flow of information among the spokes. The spokes can trade information among themselves through

the Hub. Spokes and hub can either be on the same machine or distributed. This is the environment that supports NACs. This environment embeds the *mediator* that handles the coordination model functionality and is implemented by the DARPA Communicator Architectural platform.

DARPA Communicator [9], based on MIT's Galaxy architecture is an open source hub-and-spoke architecture that provides a distributed, scriptable message passing system with special emphasis on building language-enabled dialogue systems. A Hub, implemented in C, mediates connections between Communicator servers (such as speech recognition and synthesis, parsing, dialogue management, etc.). The distribution includes server libraries for constructing Communicator-compliant servers in C (and C++), Java, Python, and Allegro Common Lisp [11].

DARPA Communicator's coordination is based on a *hub script* that specifies which servers (components) should be activated, according to matching rules. There are important semantic differences between the coordination model described in this paper and the model implemented by the DARPA Communicator. The hub is based on a fixed (and predefined) set of servers that are activated as messages posted to the hub match certain patterns. Our model requires that certain messages be delivered to multiple active components, and furthermore, that these active components may dynamically vary throughout an interaction. To compensate for these differences, we introduced functionality in the *coupler* that provides for dynamic multicasting and selective delivery.

Coupler Is the component that binds these two distinct environments together - it translates between message formats and is responsible for: 1) relaying collaboration events to the multi-agent environment for analysis and 2) propagating messages originated at the multi-agent environment among those components whose signatures comply to the messages.

The latter functionality complements the hub's by providing the (conceptually equivalent) tuple-space message distribution mechanism. For efficiency reasons, this mechanism is based on message push, rather than on a database that is polled by components, and is based on the programming style that is employed in applications, which makes easy to map messages to components. The result is conceptually equivalent to the described tuple-space based mechanism, even if perhaps less flexible.

Coupler promotes the illusion, necessary in a perceptual interface based system, that responses originated by NACs are being generated by regular participants. Coupler does that by providing a "virtual station" capability through which NACs can command the execution of actions that modify the state of one or more participants' interfaces as if they came from some other human, thus providing the uniform, low impedance contact between system and humans which is at the foundation of a perceptual user interface.

About ten thousand lines of code (mainly C/C++) implement the connection, translation between environments, as well as a highly abstracted API that is used to develop application layer components.

The operational environment involves a variety of operating systems: Linux (running DARPA Communicator), Windows 2000 (running DC-MeetingServer) and Windows XP (on the workstations).

Multimodal support in this initial phase consists of console i/o (monitor, keyboard, mouse) as well as natural language through typed and spoken messages. Natural language text output and animation, including voice production can be employed as output modalities, besides the activation of conventional widgets. Natural language processing capabilities running on the back-end are provided by language processing modules produced by Colorado University's Center for Spoken Language Research (CSLR) under the CU Communicator Project [21]. The open source CU Communicator system, is a DARPA Hub compliant system [53]. Robustness and portability of spoken dialog systems are two of the issues addressed in the project.

Currently, conventional interface components are developed in Visual Basic. Speech-to-text is built on top of SAPI (Speech API). A variety of speech-to-text engines are compliant with SAPI. We currently employ IBM's ViaVoice 9.0's engine [10]. Animation is currently built using Haptek's VirtualFriends [23].

White board, file transfer and application sharing, audio and video communication are provided directly by the H.323 infrastructure functionality.

4.6 Proof-of-concept application

A proof-of-concept application layer has been developed to validate the platform. This application includes the following interface components: a *Chat* for textual messages, an *Agenda* that registers topics and keeps track of time, a *Speak queue* that handles requests for talking, a *Mood* tool through which participants can anonymously register their emotions (bored, confused, etc.).

A simple NAC illustrates context-aware reaction. This NAC monitors the clicks on the Mood NIC instances that are active at the different participants' workstations, and either produces a private message to a participant letting her know that her feeling (e.g. bored) is not shared by other participants or by suggesting to all participants taking a break, depending on what the majority of participants has expressed over a period of time. These messages are delivered through two virtual participants (Kwebena and Kwaku), whose animated characters are displayed on participants' stations.

A Wizard of Oz interface to the animated characters allows them to be controlled remotely, basically by having them say a strings typed through the Wizard interface. These messages can be directed either to the whole group or to sub-groups or individual participants.

Experience developing these components shows that the platform does indeed support a rapid application development cycle that was expected and allows for consistency of the shared interface elements, dynamic context dependent system reaction and multimodal support, in tune with the goals of the project.

4.7 Currently implemented aspects

The introduction of common CSCW cross-cutting concerns is directly supported by the aspect-oriented use of the coordination model, as discussed in Section 4.3, can be easily composed into applications by weaving in additional service activations at appropriate *join points* (in Neem, potentially at any message received by the mediator).

Here we just briefly describe how some key aspects are handled in the current version of the platform.

- Data distribution - a distributed model is employed by the current application, i.e., each component is responsible for keeping a private copy of the data space and to update it.
- Consistency control - is guaranteed by the message serialization that takes place at the coordination mediator. Changes to the local data space replicas are applied only as a response to messages that are received from the mediator, even in cases where the action that caused the change is local. Even though it guarantees consistency, this approach suffers from potential latency problems that are typical of centralized serialization.
- Interface Coupling - a low coupling is currently employed. The results of user actions are broadcast to other interface components so that they can update their state, but there is no effort at this point to make users aware of mouse moves, detailed widget selection and operation, other than on application sharing mode, provided by the underlying H.323 infrastructure.
- Access control - different roles may operate different sets of interface elements, which might be useful for protecting information in the sense that other participants would lack the tools to access this information. No other provisions are considered at this point, and in particular, no protection against malicious users is implemented.
- Concurrency control - a social protocol is assumed as the resolution method for user conflicts at this point.

Alternative ways of handling these aspects will be developed to meet specific application requirements.

5 Summary and future work

The Neem Platform is a research test bed for University of Colorado's Project Neem, which is concerned with the development of socially and culturally aware group collaboration systems. The use of perceptual interfaces is a cornerstone of this project. One of the research hypotheses of the project is that social mediation systems can benefit from the low impedance provided by perceptual interfaces, that blur to some extent the distinction between human and system participation in an interaction.

The Neem Platform is a generic framework for the development of collaborative applications. It supports rapid development of augmented real-time distributed multipoint group applications that employ a perceptual interface paradigm, based on how humans communicate among themselves.

The platform is an extensible, open-ended, application neutral infrastructure that offers communication services necessary to integrate new functionality on two levels: 1) Integration of new platform functionality, such as new devices/modalities and 2) Development of specific applications. The platform thus offers a mechanism for reuse of generic functionality and supports rapid application development by encapsulating communications details.

Neem's coordination model is based on a pair of tuple-spaces to which components are allowed to write to and read from, respectively, and mediation functionality that is driven by reified composition rules that specify transformation of messages into zero or more messages that in turn activate potentially multiple services provided by components.

We showed that this model can be usefully employed to support an Aspect-Oriented development paradigm. Cross-cutting concerns are common in collaborative applications, and each class of applications might require adaptations to one or more flavors of these aspects.

A prototype of the platform has been implemented. Actual development leverages as much as possible on existing, field tested technology, such as H.323 MCUs, the DARPA Communicator architecture and commercial speech and animation engines.

Future work will enhance the platforms capabilities by expanding its multimodal functionality, including facial analysis, gestures, including American Sign Language capabilities in a robust way.

Different applications are being developed and deal with the challenging aspects of building culturally and socially adequate tools and virtual participants. Planned applications include business meetings and distance education applications.

References

1. J. Andreoli, H. Gallaire, and R. Pareschi. Rule-based object coordination. In P. Ciancarini, O. Nierstrasz, and A. Yonezawa, editors, *ObjectBased Models and Languages for Concurrent Systems*, volume 924 of *Lecture Notes in Computer Science*, pages 1–13, Berlin, 1995. Springer-Verlag.
2. J.-P. Banâtre and D. Le Mètayer. Gamma and the chemical reaction model: ten years after. In J.-M. Andreoli, H. Gallaire, and D. Le Mètayer, editors, *Coordination programming: mechanisms, models and semantics*, pages 1–39, 1996.
3. R. A. Bolt. "put-that-there": Voice and gesture at the graphics interface. In *SIGGRAPH '80 Proceedings*, volume 14, pages 262 – 270, July 1980.
4. N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, April 1989.
5. S. Castellani and P. Ciancarini. Enhancing coordination and modularity mechanisms for a language with objects-as-multisets. In P. Ciancarini and C. Hankin, editors, *Coordination Languages and Models*, volume 1061 of *Lecture Notes in Computer Science*, pages 89–106. Springer-Verlag, 1996.
6. S. Castellani, P. Ciancarini, and D. Rossi. The shape of shade: a coordination system. Technical Report UBLCS 96-5, 1996.
7. E. Clergue, M. Goldberg, N. Madrane, and B. Merialdo. Automatic face and gestural recognition for video indexing. In Martin Bichsel, editor, *International Workshop on Automatic Face- and Gesture-Recognition*, pages 110–115, Zurich, Switzerland, June 1995.
8. P. R. Cohen, M. Johnston, D. McGee, I. Smith, S. Oviatt, J. Pittman, L. Chen, and J. Clow. Quickset: Multimodal interaction for simulation set-up and control. In *Proceedings of the Fifth Applied Natural Language Processing meeting*, Washington, D.C., 1997.
9. MITRE Corp. http://fofoca.mitre.org.
10. IBM Corporation. Ibm voice systems. http://www-4.ibm.com/software/speech/.
11. Mitre Corporation. *Galaxy Communicator Documentation*. Mitre Corporation. Available on the web at http://fofoca.mitre.org/sites/MITRE/Galaxy-public/docs/index.html.

12. Inc Data Connection. http://www.dataconnection.com.
13. Prasun Dewan. Architectures for collaborative applications. *Trends in Software, special issue on Computer Supported Cooperative Work*, 7:169–194, 1998.
14. Paul Dourish. Consistency guarantees: Exploiting application semantics for consistency management in a collaboration toolkit. In *Proceedings of ACM CSCW'96 Conference on Computer-Supported Cooperative Work*, Concurrency, pages 268–277, 1996.
15. Paul Dourish. Using metalevel techniques in a flexible toolkit for CSCW applications. *ACM Transactions on Computer-Human Interaction*, 5(2):109–155, 1998.
16. Paul Dourish and W. Keith Edwards. A tale of two toolkits: Relating infrastructure and use in flexible CSCW toolkits. *Journal on Computer Supported Cooperative Work*, 9(1), 2000. Special issue on Tailorable Systems and Cooperative Work.
17. W. Keith Edwards. *Coordination Infrastructure in Collaborative Systems*. PhD thesis, Georgia Institute of Technology, College of Computing, Atlanta, GA, December 1995.
18. C.A. Ellis. Neem project: An agent based meetings augmentation system. Technical report, University Of Colorado at Boulder, Department of Computer Science, March 2002. http://www.cs.colorado.edu/~skip/NeemAgents.pdf.
19. Tzilla Elrad, Robert E. Filman, and Atef Bader. Aspect-oriented programming: Introduction. *Communications of the ACM*, 44(10):29–32, October 2001.
20. Tzilla Elrad, Robert E. Filman, and Atef Bader, editors. *Communications of the ACM. Special Issue on Aspect-Oriented Programming*, volume 44, 2001.
21. Colorado University Center for Spoken Language Research. Cu communicator page. http://communicator.colorado.edu/.
22. S. Greenberg and M. Roseman. Groupware toolkits for synchronous work. Technical Report 96-589-09, Department of Computer Science, University of Calgary, 1996.
23. Haptek. http://www.haptek.com.
24. Jakob Hummes and Bernard Merialdo. Design of extensible component-based groupware. *Journal on Computer Supported Cooperative Work*, 9(1):53–74, 2000. Special issue on Tailorable Systems and Cooperative Work.
25. Katherine Isbister, Cliff Nass, Hideyuki Nakanishi, and Toru Ishida. Helper agent: An assistant for human-human interaction in a virtual meeting space. In *Proceedings of the CHI 2000 Conference*, 2000.
26. Tony Jebara, Yuri Ivanov, Ali Rahimi, and Alex Pentland. Tracking conversational context for machine mediation of human discourse. In *AAAI Fall 2000 Symposium - Socially Intelligent Agents - The Human in the Loop*, November 2000.
27. Michael Johnston, Philip R. Cohen, David McGee, Sharon L. Oviatt, James A. Pittman, and Ira Smith. Unification-based multimodal integration. In *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, 1997. http://citeseer.nj.nec.com/135215.html.
28. Gregor Kiczales. Beyond the black box: open implementation — soapbox. *IEEE Software*, 13(1), January 1996.
29. Michael Koch and Gunnar Teege. Support for tailoring cscw systems: Adaptation by composition. In *Proc. 7th Euromicro Workshop on Parallel and Distributed Processing*, pages 146–152, Funchal, Portugal, February 1999. IEEE Press. http://www11.in.tum.de/publications/pdf/Koch1999a.pdf.
30. D.B. Koons, C.J. Sparrell, and K.R. Thorisson. Integrating simultaneous input from speech, gaze, and hand gestures. In M. Maybury, editor, *Intelligent Multimedia Interfaces*, chapter 11, pages 252–276. MIT Press, Menlo Park, CA, 1993.
31. T.W. Malone, K.-Y. Lai, and C. Fry. Experiments with oval: A radically tailorable tool for cooperative work. *ACM Transactions on Information Systems*, 13(2):177–205, 1995.

32. Gloria Mark. Extreme collaboration. *Communications of the ACM*, 2002. forthcoming. http://www1.ics.uci.edu/ gmark/Mark-CACM-final.pdf.
33. D. Martin, A. Cheyer, and D. Moran. The Open Agent Architecture: a framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1/2):91–128, 1999.
34. D. McGee and P. Cohen. Creating tangible interfaces by augmenting physical objects with multimodal language. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI 2001)*, pages 113–119, Santa Fe, NM, January 2001.
35. Laurence Nigay and Joëlle Coutaz. A design space for multimodal systems: Concurrent processing and data fusion. In *Proceedings of INTERCHI '93*, 1993. http://citeseer.nj.nec.com/nigay93design.html.
36. Kazushi Nishimoto, Yasuyuki Sumi, and Kenji Mase. Enhancement of creative aspects of a daily conversation with a topic development agent. In *Coordination Technology for Collaborative Applications – Organizations, Processes, and Agents*, volume 1364 of *Lecture Notes on Computer Science*, pages 63–76. Springer-Verlag, 1998.
37. S.L. Oviatt and P.R. Cohen. Multimodal systems that process what comes naturally. *Communications of the ACM*, 43(3):65–70, March 2000. http://www.acm.org/pubs/citations/journals/cacm/2000-43-3/p45-oviatt/.
38. George A. Papadopoulos and Farhad Arbab. Coordination models and languages. In *The Engineering of Large Systems*, volume 46 of *Advances in Computers*. Academic Press, August 1998.
39. E.D. Petajan. Automatic lipreading to enhance speech recognition. In *Proceedings of the IEEE Communication Society Global Telecommunications Conference*, November 1984.
40. Janus Project. http://www.is.cs.cmu.edu/mie/janus.html.
41. Rameshsharma Ramloll. Micis: A multimodal interface for a common information space. In *ECSCW'97 Conference Supplement*, Lancaster, UK, September 1997.
42. Byron Reeves and Clifford Nass. *The Media Equation. How People Treat Computers, Television, and New Media Like Real People and Places.* CSLI Publications, Stanford, CA, 1996.
43. Byron Reeves and Clifford Nass. Perceptual user interfaces: perceptual bandwidth. *Communications of the ACM*, 43(3):65–70, March 2000.
44. Henry M. Robert. *Roberts Rules of Order Revised for Deliberative Assemblies*. Scott, Foresman, Chicago, 1915. Online edition at http://www.bartleby.com/176/.
45. Mark Roseman and Saul Greenberg. Building real-time groupware with groupkit, a groupware toolkit. *ACM Transactions on Computer-Human Interaction*, 3(1):66–106, 1996.
46. Kjeld Schmidt and Carla Simone. Coordination mechanisms: Towards a conceptual foundation of CSCW systems design. *Computer Supported Cooperative Work*, 5(2/3):155–200, 1996.
47. Timo Sowa, Martin Frohlich, and Marc Erich Latoschik. Temporal symbolic integration applied to a multimodal system using gestures and speech. In A. Braffort et al., editor, *Toward a Gesture-based Communication in Human-Computer Interaction - Proceedings of the International Gesture Workshop*, pages 291–302, Gif-sur-Yvette, France, March 1999. Springer-Verlag (LNAI). http://www.techfak.uni-bielefeld.de/˜tsowa/download/Gif-sur-Yvette.pdf.
48. Oliver Stiemerling and Armin B. Cremers. The evolve project: Component-based tailorability for cscw applications. *AI & Society*, 14:120–141, 2000. http://citeseer.nj.nec.com/294373.html.
49. OAA Development Team. Open agent architecture: Technical white paper. Technical report, SRI International, 1999. http://www.ai.sri.com/ oaa/whitepaper.html.
50. Robert Tolksdorf. Laura: A coordination language for open distributed systems. In *International Conference on Distributed Computing Systems*, pages 39–46, 1993.
51. M. Turk and G. Robertson, editors. volume 43 of *Communications of the ACM*, March 2000. Special Issue on Perceptual User Interfaces.

52. International Telecommunication Union. http://www.itu.org.
53. Ward W. and B. Pellom. The cu communicator system. In *IEEE Workshop on Automatic Speech Recognition and Understanding*, Keystone Colorado, December 1999.
54. Alex Waibel. Interactive translation of conversational speech. *IEEE Computer*, 29(7):41–48, July 1996.